Introdução à programação em Python

Eduardo Adame Salles
Paulo Victor dos Santos

8 de Outubro de 2020

Conteúdo

1 Introdução ao Curso

1.1 O que é Python

Python é uma linguagem de programação muito utilizada nos dias atuais. Ela tem como um dos maiores princípios a legibilidade e a facilidade do código. Com ele, é possível criar desde simples aplicações baseadas em texto, até programas complexos com interfaces gráficas.

Principais Vantagens:

- Sintaxe simples
- Multiplataforma
- Grande comunidade e documentação
- Muitas expansões (módulos)

1.2 Onde Python foi utilizado

Existem muitos softwares e projetos que têm como tecnologia principal o Python. Por exemplo, o computador de placa única Raspberry Pi teve como propósito inicial vibilizar o ensino de Python nas escolas. No caso de exemplos de softwares, podemos listar:

- Youtube
- Instagram
- Dropbox
- Pinterest
- Spotify
- Reddit

1.3 Como o Python interpreta os códigos

É muito simples, veja: Um interpretador chamado `python` (que deve ser instalado no Windows e já vem instalado no Linux e MacOS) recebe como entrada um arquivo de texto terminado em `.py` e realiza as operações lógicas e matemáticas de cada linha de cima para baixo. Linhas começadas por `#` não são interpretadas (comentários).

Para escrever esse arquivo `.py` usamos um outro programa chamado IDE. Existem vários pela Internet, como o Pycharm, Thonny, Atom, IDLE e Vscode. Neste curso, usaremos o Vscode.

2 Começando a programar

2.1 Seu primeiro programa em Python

Para vocês verem como é simples programar em Python, vamos escrever seu primeiro programa em uma única linha.

```
1 print('Olá mundo')
```

Em um primeiro momento, não precisa se atentar aos significado de `print()`, mas você fez um programa que imprime o texto *Olá Mundo*. Tente trocar o que está entre as aspas e compile novamente

Para artigo de comparação, veja o mesmo programa escrito em outra linguagem de programação, como Java:

```
1 public class Java {
2         public static void main(String[] args) {
3                 System.out.println("Olá Mundo");
4         }
5 }
```

Claramente muito mais complicado que o Python.

2.2 Variáveis

Para armazenar valores na programação, nós utilizamos variáveis.

Toda variável tem um tipo, que determina como ela se comportará em operações matemáticas e lógicas. Os principais tipos de variáveis são:

- Integers (Valores numéricos inteiros)
- Floats (Valores numéricos com casas decimais)
- Strings (Textos, palavras, etc)
- Booleans (True, False)

- Lists (Listas com valores e/ou variáveis)

Para definir uma variável, é muito simples. Basta dar um nome a ela (sem espaços) e utilizar o sinal de igual com o valor em seguida. O tipo é determinado automaticamente pelo compilador com base na sintaxe do valor agregado.

Para Integers, utilize o número normalmente:

```
meuInteiro = 10
```

Para Floats, utilize o número normalmente com as casas decimais separadas por ponto:

```
meuFlutuante = 10.015
```

Para Strings, escreva entre aspas simples ou duplas:

```
meuNome = 'Eduardo'
meuSobrenome = "Adame"
```

Para Booleans, utilize `True` ou `False`

```
euGostoDePython = True
euGostoDeLiteratura = False
```

Por fim, para Lists, basta criar um espaço delimitado por colchetes `[ ... ]` e separar os valores por vírgulas. Voltaremos a falar mais sobre esse tipo mais tarde.

```
meusDados = ['Eduardo',17,1.78]
```

2.3 Recebendo Inputs

Assim como a função `print()` imprime o que está dentro dos parênteses a função `input()` faz o mesmo, mas também exibe um campo para entrada de texto por parte do usuário, que será do tipo String.

Porém, veja o seguinte exemplo:

```
input('Qual o seu nome? ')
```

Ao fazer isso, o valor recebido pelo `input()` não será armazenado em lugar nenhum. Para que isso aconteça, precisamos agregar essa função à uma variável. Veja:

```
nome = input('Qual o seu nome? ')
```

2.4 Utilizando variáveis

Para utilizar o valor de uma variável basta utilizar o nome que você deu a ela. Por exemplo, seguindo exemplos anteriores, `print(meuInteiro)` vai imprimir 10, e `print(Nome)` vai imprimir o nome que você escrever.

Agora, se você quiser imprimir uma mensagem como: "Seu nome é ..." utilizando uma variável no local dos pontos você deve utilizar Strings formatadas.

Dê uma olhada nesse programa:

```
nome = input('Qual o seu nome? ')
print('Seu nome é ', nome)
print('Seu nome é {}'.format(nome))
print(f'Seu nome é {nome}')
```

A mensagem será impressa igualmente 3 vezes, pois as três formas, nesse caso, são válidas. A primeira é a mais simples, onde você concatena a variável à String; a segunda é um exemplo de formatação, a variável dentro de `.format()` ficará no lugar de `{}`; a última é uma F-String, você coloca um f antes das áspas e a variável entre chaves.

Desafio: Criar um programa que imprima a mensagem: "Seu nome é NOME, e seu sobrenome é SOBRENOME", onde NOME e SOBRENOME são entradas do usuário.

2.5 Convertendo tipos de variáveis

Como vamos falar sobre operações matemáticas na próxima seção, precisamos entender que uma String `'10'` é diferente de um Inteiro `{}`, e que, possivelmente, você irá querer utilizar ele como um número.

Para converter em String, basta colocar o valor/variável dentro da função `str()`

```
massa = 78.5
massa2 = str(massa)
```

Para converter em Inteiro, basta colocar o valor/variável dentro da função `int()`

```
idade = '17'
idade2 = int(idade)
```

Para converter em Float, basta colocar o valor/variável dentro da função `float()`

```
pi = '3.14159'
pi2 = float(pi)
```

Como dito anteriormente, inputs retornam Strings, e para receber esse valor como um número podemos converter logo na associação à uma variável.

```
idade = int(input('Quantos anos você tem?'))
print('Você tem {} anos de vida'.format(idade))
```

Todos concordam que idades são inteiros?

2.6 Operações Matemáticas

Aqui a brincadeira começa a ficar legal, em nativamente em Python nós temos as operações básicas e algumas funções para arredondar e coisas assim.

- Adição => `+`
- Subtração => `-`
- Multiplicação => `*`
- Divisão => `/`

- Potenciação => `**`
- Módulo (Resto) => `%`

Lembrando que essas operações só tem comportamentos matemáticos em Inteiros e Flutuantes. (Teste as operações com uma String e veja o que acontece)

Veja esse exemplo de programa:

```
1 a = int(input('Digite o primeiro número: '))
2 b = int(input('Digite o segundo número: '))
3 soma = a + b
4 produto = a * b
5
6 print(f'O resultado da soma dos dois é {soma} e de seu produto é {produto}')
```

Desafio: Sabendo que, quando $ax + b = 0$, $x = \frac{-b}{a}$, faça um programa que encontra o valor de x a partir da entrada de a e b pelo usuário.

2.7 Operações Lógicas

Essas operações são muito importantes para aplicar condições. O que a diferencia é que, mesmo se utilizados qualquer tipo de variável, essa operação retorna um Boolean.

As comparações mais comuns são:

- Igual => `==`
- Diferente => `!=`
- Maior (maior ou igual) => `>` `(>=)`
- Menor (menor ou igual) => `<` `(<=)`

E as operações podem ser escritas por extenso:

- Operador 'E' => `and`
- Operador 'Ou' => `or`

Por exemplo, você pode imprimir essas operações em um primeiro momento para entendê-las:

```
print(2==1)
print(2!=1)
print(1==1)
print('Eduardo' == 'eduardo')
print(10>9)
print(10<9)
print(2==2 and 1==1)
print(2==1 and 1==1)
print(2==1 or 1==1)
print(2==1 or 1==3)
```

2.8 If e Else

If e Else ("se", e "caso contrário"em português) são estruturas para avaliar condições e executar uma ação caso ela for cumprida. Por exemplo, vamos fazer um programa que vê qual número é maior, e imprime uma mensagem pra gente mostrando isso.

```
a = float(input('Digite o primeiro número: '))
b = float(input('Digite o segundo número: '))

if a>b:
        print(f'{a} é maior que {b}')
else:
        print(f'{a} é menor ou igual a {b}')
```

Aqui podemos ver como funciona a estrutura, o mais importante é entender que o Python funciona como se fosse em blocos. Toda vez que usa-se dois pontos (`:`) devemos escrever com identação (com TAB, parágrafo, etc) o que está dentro desse bloco.

No caso, temos o bloco `if` que executa um `print()` caso sua condição for cumprida e temos o bloco `else` que executa outro `print()` caso a condição do `if` anterior não for cumprida.

Ainda se quiser adicionar outras condições, é possível utilizar a partícula `elif` e `else` rodará somente se todas as condições não forem cumpridas.

Veja:

```
a = float(input('Digite o primeiro número: '))
b = float(input('Digite o segundo número: '))

if a>b:
        print(f'{a} é maior que {b}')
elif a<b:
        print(f'{b} é maior que {a}')
else:
        print(f'{a} é igual a {b}')
```

2.9 Loops

Basicamente você já consegue fazer muita coisa com o que sabe, mas falta um pontinho: loops. Loops são uma forma de repetir um bloco de código.

Existem duas formas de loop em Python, o `while` e o `for`.

No caso do `while`, o bloco de código será repetido enquanto uma condição for verdadeira, veja:

```
a = int(input('Digite um número entre 1 e 20: '))

while a!=12:
        a = int(input('Tente novamente: '))

print('Parabéns! Você acertou o número')
```

Esse é um jogo para você acertar o número definido na condição, veja que enquanto a for diferente de 12 ele vai te pedir para tentar outro número.

O outro tipo de loop, e mais complexo, `for`, tem uma outra sintaxe. Ele executa o bloco para cada item em uma lista, e pode usar esse item. Tente entender com o exemplo:

```
nomes = ['Eduardo', 'Paulo', 'Marcelo', 'Ruan', 'Diego', 'Natã',
    'Camille','Samuel',]

for nome in nomes:
        print(nome)
```

Como se pode ver, ele repete o bloco de código para cada item na lista, e utiliza esse item em sua vez, que fica armazenado em uma variável que você escolhe o nome (nesse caso foi nome também kk). O resultado desse código deve ser cada nome em linhas diferentes, mas você pode criar coisas mais complexas, como condições ou operações matemáticas.

3 Utilizando Funções e Classes

3.1 Funções

As funções são blocos de código que rodam quando chamados. Eles podem ter parâmetros, ou não. É uma forma muito boa de organizar e otimizar seu programa, de forma que não se repita códigos e códigos.

Para definir uma função é simples:

```
def imprime_meu_nome():
        print('Meu nome é Eduardo')

imprime_meu_nome()
```

Essa função é muito simples, e não tem muita utilidade. Mas, primeiro ela é definida, e depois invocada. Para ter mais utilizade, veja o que pode ser feito com parâmetros:

```
def imprime_nome(nome):
        print(f'Seu nome é {nome}')

imprime_nome('Eduardo')
imprime_nome('Pedro')
```

Essa função, após definida, é invocada com dois parâmetros diferentes, sem ter que escrever tudo novamente. Imagine ter que escrever 20 linhas para cada variável que usar.

Também é possível utilizar mais de um parâmetro:

```
1 def imprime_dados(nome, idade):
2         print(f'{nome} tem {idade} anos de vida')
3
4 imprime_dados('Eduardo', 17)
5 imprime_dados('Pedro', 20)
```

3.1.1 Return

A partícula `return` é muito importante. Ela dá o valor retornado pela função. Por exemplo, se você tentar imprimir uma função que não retorna nada, você não imprimirá nada. Vamos utilizar o exemplo anterior de forma que só utilizemos a função para formar a frase, e imprimimos manualmente depois.

```
1 def dados(nome, idade):
2         return f'{nome} tem {idade} anos de vida'
3
4 print(dados('Eduardo', 17))
5 print(dados('Pedro', 20) + ', mas ainda é infantil' )
```

Você pode usar essa estrutura para retornar o que quiser, até outra função.

3.2 Classes

As classes são muito interessantes. Uma classe é um forma de criar um modelo para ser utilizado, que pode possuir funções e variáveis próprias. Por exemplo, podemos criar a classe Aluno:

```
class Aluno:
        def __init__(self, nm, ide, dsc):
                self.nome = nm
                self.idade = ide
                self.disciplina = dsc
        def estudar(self):
                print(f'{self.nome} estudou {self.disciplina}')
        def carteirinha(self):
                print(f'Nome:{self.nome} \n Idade:{self.idade}')
```

E após criar essa classe, podemos simplesmente criar alunos, como criamos variáveis. Note que `__init__(self)` é a função construtuora e é chamada sempre que um objeto é criado, `self` representa o objeto criado a partir dessa classe.

```
class Aluno:
        def __init__(self, nm, ide, dsc):
                self.nome = nm
                self.idade = ide
                self.disciplina = dsc
        def estudar(self):
                print(f'{self.nome} estudou {self.disciplina}')
        def carteirinha(self):
                print(f'Nome:{self.nome} \n Idade:{self.idade}')

eduardo = Aluno('Eduardo', 17, 'Tratamentos Térmicos')

print(eduardo.nome)
eduardo.estudar()
eduardo.carteirinha()
```

Não gostaria de deixar de mencionar a partícula `pass` que quando utilizada dentro de uma classe ou função, a classe ou função é interrompida. O mesmo acontece ao utilizar o `break` em um `while` loop.